CS422 - Final Exam

Time: until 23:59, Thursday Dec 15. Send it either by email or push it under my door (SC 2110).
Total: 100 (out of 105) points needed for maximum credit.
You are *not* allowed to collaborate or discuss these problems with each other. You can use books, lecture notes, coffee, etc.

Problem 1. (15 points)
Consider programs of the following form, written in an uncurried `FUN`-like language supporting the various parameter passing styles discussed in class:

```
let f(P x, P y) = x * y
and g(Q x, Q y) = {
                      x := x + y ;
                      y := x - y ;
                      x := x - y ;
                      x + y
                  }
and x = a and y = b
in f(g(x,y), g(y,y))
```

Here `a` and `b` are some arbitrary integers, assumed given. `P` and `Q` stay for parameter passing styles. What values do these programs evaluate to, when `P` is any of call-by-value, call-by-name, or call-by-need, and when `Q` is any of call-by-reference or call-by-value-result? For each of the six situations, explain how you obtained the result.

Problem 2. (15 points)
Parallel assignments, typically written `X1, X2, ..., Xn := E1, E2, ..., En`, can be a useful feature of a programming language, because they allow one to write more compact, abstract and intuitive code. A parallel assignment first evaluates the expressions in its right-hand-side *in parallel* and then assigns the obtained values to the names listed in its left-hand-side, in the corresponding order. For example, `x,y := y,x` swaps the values of `x` and `y` without a need for a temporary variable. Suppose that you, as a language designer, want to add such parallel assignments to your programming language, in this case `FUN`. Also, suppose that you want to take full advantage of a highly parallel rewrite engine, in the sense of evaluating the `n` expressions indeed in parallel, as if they are different execution threads. This problem has two parts:

1. Show that the parallel assignment feature is just "syntactic sugar", in the sense that it can be automatically translated into an equivalent expression using only the already existing `FUN` language features;

2. Give the parallel assignment a direct continuation-based semantics using `K`.

You do not need to redefine anything that was already defined in the lecture notes.

Problem 3. (15 points)
Suppose that you execute the object oriented program below:

```
class A {
  field value;
  method initialize(v) { value := v }
  method m() { value + 10 }
}
class B inherits A {
  method m() { super m() }
}
class C inherits B {
  field obj;
  method void initialize(v,o) {
    super initialize(v);
    obj := o
  }
  method m() { value + send obj m() }
}
class D inherits B {
  method initialize(v) { super initialize(v) }
  method m() { value * 2 }
}
main
  let b = ...
  in let o = if b then new C(5, new C(10, new D(15))) else new D(20)
     in send o m()
```

What value will be returned under dynamic method invocation when b is true? When b is false? What value will be returned under static method invocation when b is true? When b is false? Assume we are in a typed environment, even without typing annotations, so you should take the types of expressions into account. Justify your answers.

Problem 4. (30 points)

(7 points) Write a FUN program that calculates all the k-combinations of n elements, that is, all the possibilities to pick k elements from a pool of n elements. It is known that there are $n!/(k! \cdot (n-k)!)$ such combinations; you are supposed to define a function taking two arguments n and k and returning a list of lists of numbers between 1 and n; these lists are regarded as sets (no repetitions and the order of elements does not matter).

(10 points) Type the program above by hand using the type inference technique discussed in class. Give enough detail to show that you understand the technique, but not more than that.

(13 points) Apply the CPS transformation to the program above by hand using the technique discussed in class. Again, give enough detail to show that you understand the CPS transformation technique.

Problem 5. (15 points)
Prove the following partial correctness assertion using the Hoare logic inference rules:

```
{x == m and y == n and n >= 1 and z == 1}
  while (y >= 1) (
    while (even?(y)) (
      x = x * x ;
      y = y / 2) ;
    z = z * x ;
    y = y - 1)
{z = m^n}
```

where `m^n` is the power of `m` to the `n` and `even?(y)` tests whether `y` is an even number. As usual, do all the state assertion validity computations by hand and only show the applications of the Hoare inference rules.

Problem 6. (15 points)
Write the dining philosophers program in FUN (with concurrency). You can assume just 3 philosophers and you can assume that each acquires first the left fork, then the right one, then eat (don't need to do anything here, just put a comment in the code saying "eating"), then release the right fork and then the left one. The philosophers must be threads and the forks must be locks. Say how many states this 3-threaded program can possibly reach, where states are counted only when rewrite rules apply (the only rewrite rule that you need here is the one for acquiring a lock). You are allowed to use Maude's search capabilities to count the number of states, though you do not need or have to. How can you say when the program is deadlocked? How many deadlock states are there? Is ordinary testing powerful enough to detect this deadlock?